



# University of Maryland College Park

## Department of Computer Science

### CMSC132 Fall 2022

#### Exam #2

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g., 123456789):

#### Instructions

- Please print your answers and use a pencil.
- This exam is a closed-book, closed-notes exam with a duration of 50 minutes and 100 total points.
- **Do not remove the exam's staple.** Removing it will interfere with the scanning process (even if you staple the exam again).
- Write your directory id (e.g., terps1, not UID) at the bottom of pages with **DirectoryId**.
- Provide answers in the rectangular areas.
- Do not remove any exam pages. Even if you don't use the extra pages for scratch work, return them with the rest of the exam.
- Your code must be efficient and as short as possible.
- If you continue a problem on the extra page(s) provided, make a note on the particular problem.
- You don't need to use meaningful variable names; however, we expect good indentation.
- **You must write your name and id at this point (we will not wait for you after time is up).**
- You must stop writing once time is up.

#### Grader Use Only

#1	Part #1 (Short Answer)	10	
#2	Part #2 (Code 1)	20	
#3	Part #3 (Code 2)	70	
<b>Total</b>	Total	100	

## Part #1 (Short answer – 2 points each)

1. List the following Big O expressions in order of asymptotic complexity (lowest complexity first).

$O(n \log(n))$

$O(n^3)$

$O(\log(n))$

$O(n^2)$

$O(n)$

2. Assume you consider an algorithm for a given task to be fast if it is in  $O(n \log(n))$ . You are given an algorithm in  $O(n)$ . Referencing the definition of Big O, explain why the given algorithm will be fast. No more than 2 sentences.

3. Indicate the algorithm complexity of the following expression using the best bound:  $70n + 2n \log(n) + n^3$

4. Invocation of the default implementation of `clone` method makes this type of copy? \_\_\_\_\_
5. Term used to denote the feature found in Java that deallocates objects in the heap that can no longer be reached via an available reference? \_\_\_\_\_

**The code given below will be used in both Part 2 and Part 3. All given code and code you write will be in the same package.**

<pre>public abstract class Animal {     private String name;     public Animal(String name) {         this.name = name;     }     public String getName() {         return name;     }     @Override     public abstract String toString(); }</pre>	<pre>public class Bird extends Animal{     public Bird(String name) {         super(name);     }     @Override     public String toString() {         return getName() + " ";     } }</pre>	<pre>public class Mammal extends Animal{     public Mammal(String name) {         super(name);     }     @Override     public String toString() {         return getName() + " ";     } }</pre>
---	---	---

## Part #2 (Code 1)

1. Using your knowledge of bounded wildcards, write the code for the static method `longestName` that simply returns the name of the animal in the list passed in with the most number of characters. It should have one `ArrayList` parameter with a bound that must be one of the 3 classes above. Use the `length` method of the string to get the length of the name. Assuming there is more than one animal with the longest name in the list, simply return the name of the first one encountered when traversing the list from the start.

```
public class AnimalName {
```

```
    }

    public static void main(String[] args) {
        ArrayList<Bird> bList= new ArrayList<Bird>();
        ArrayList<Mammal> mList= new ArrayList<Mammal>();
        ArrayList<Animal> aList= new ArrayList<Animal>();

        bList.add(new Bird("Parrot")); bList.add(new Bird("Vulture"));bList.add(new Bird("Hawk"));
        mList.add(new Mammal("Tiger"));mList.add(new Mammal("Lion"));mList.add(new Mammal("Fox"));
        aList.addAll(bList);  aList.addAll(mList);

        System.out.println(longestName(bList)); //prints Vulture
        System.out.println(longestName(mList)); //prints Tiger
        System.out.println(longestName(aList)); //prints Vulture
    }
}
```

Directory ID:

## Part #3 (Code 2)

Given the code below, finish the 3 missing methods: `sortRoster`, `clone`, and `iterator`

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.NoSuchElementException;

public class ZooRoster implements Iterable<Bird>, Cloneable {

    private ArrayList <Animal> roster;

    public ZooRoster(ArrayList<Animal> roster) { //assume roster not null
        this.roster = new ArrayList<Animal>(roster);
    }

    public void add(Animal a){
        roster.add(a);
    }

    @Override
    public String toString() {
        return roster.toString()+"\n";
    }

    public void sortRoster(){
        Collections.sort(roster, ?); //substitute your lambda expression for ?
    }

    @Override
    public ZooRoster clone() {
        //write the code, the clone should be independent of the original
    }

    @Override
    public Iterator<Bird> iterator() {
        //write the code using local class syntax that returns an iterator supporting next and hasNext
    }
}
```

### Driver Output

```
[Parrot , Hawk , Tiger , Lion , Owl , Bear , Vulture , Fox , Penguin , Bigfoot ]
[Parrot , Hawk , Tiger , Lion , Owl , Bear , Vulture , Fox , Penguin ]
[Hawk , Owl , Parrot , Penguin , Vulture , Bear , Fox , Lion , Tiger ]
Parrot Hawk Owl Vulture Penguin
Did not call hasNext
true
Parrot
true
Hawk
true
Owl
true
Vulture
Did not call hasNext
true
Penguin
false
No more elements
```

## Driver

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.NoSuchElementException;

public class SampleDriver {

    public static void main(String[] args) {

        String result = "";

        ArrayList <Animal> roster= new ArrayList <Animal>();
        roster.add(new Bird("Parrot"));roster.add(new Bird("Hawk"));roster.add(new Mammal("Tiger"));
        roster.add(new Mammal("Lion"));roster.add(new Bird("Owl"));roster.add(new Mammal("Bear"));
        roster.add(new Bird("Vulture"));roster.add(new Mammal("Fox"));roster.add(new Bird("Penguin"));

        ZooRoster zList = new ZooRoster(roster);
        ZooRoster zListOther = zList.clone();

        zList.add(new Mammal("Bigfoot")); //probably would be a mammal :)

        result += zList; //Bigfoot in here
        result += zListOther; //but not in clone

        zListOther.sortRoster();
        result += zListOther; //sorted

        for(Bird b: zList){ //back to the original list
            result +=b; //only the birds
        }
        result += "\n";

        Iterator<Bird> bIt = zList.iterator(); //new Iterator

        try {
            result+= bIt.next()+ "\n"; //no next if not called hasNext
        }
        catch (UnsupportedOperationException e){
            result+= e.getMessage()+"\n";
        }

        for (int i =0; i <4; i++){ //the first 4 birds one at a time
            result+= bIt.hasNext()+ "\n";
            result+= bIt.next()+ "\n";
        }

        try {
            result+= bIt.next()+ "\n"; //calling next twice
        }
        catch (UnsupportedOperationException e){
            result+= e.getMessage()+"\n";
        }

        result+= bIt.hasNext()+ "\n"; //last bird from original list
        result+= bIt.next()+ "\n";

        result+= bIt.hasNext()+ "\n"; //nothing left
        try {
            result+= bIt.next()+ "\n";
        }
        catch (NoSuchElementException e){
            result+= e.getMessage()+"\n";
        }

        System.out.println(result);
    }
}
```

Directory ID:

1. Write the code for the clone method, so that the cloned object will be independent of the original. You can assume that the Animal, Bird, and Mammal class will remain immutable.

```
public ZooRoster clone() {
```

2. Simply write the code that would get substituted for the second argument (i.e. ?) as a **lambda expression**.

```
public void sortRoster(){
    Collections.sort(roster, ?); //substitute your lambda expression for ?
}
```

The second argument is simply the implementation of the `compare` method as found in the functional interface `Comparator`. From the Java API: `int compare(T o1, T o2)` - Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second. Code it so that any bird is "less than" a mammal and if the 2 animals are the same type, comparison is based on the name (just use `compareTo` of the `String`).

3. Write the code for the `iterator` method so that the `ZooRoster` class is an `Iterable<Bird>`. This means you have to code up the `next` and the `hasNext`, and when you iterate over the `ZooRoster` object only the birds are returned by `next`.
- **You must use local class syntax** (not anonymous) and can call your local class whatever you want.
  - Your local class should have a `public boolean hasNext()` and a `public Bird next()`, and return an instance of the local class from the `iterator` method.
  - You can have as many initialized fields as you want in your local class, but no other methods besides `next` and the `hasNext`. This also means no explicit constructor or initialization blocks.
  - Your implementation must enforce that the `hasNext` has been called prior to each call of `next`. If not, throw a `new UnsupportedOperationException("Did not call hasNext");`
  - Once the iteration is complete (i.e. `hasNext` returns `false`) any call to `next` will throw a `new NoSuchElementException("No more elements");`
  - As for code from the Java library, you can use the following 2 `ArrayList` methods:

`E get(int index)` - Returns the element at the specified position in this list.  
`int size()` - Returns the number of elements in this list.

```
public Iterator<Bird> iterator() {
```



**EXTRA PAGE IN CASE YOU NEED IT**

**LAST PAGE**